Lab 10: Arrays

Taaruni Ananya | Session 008 | CRN: 15623

04/04/2025

Dr. Lan Gao

Introduction

Lab 10 is centered around programming for taking an array as input and returning an output, which involves practicing writing code with RISC-V, SNOWBALL, and Terminal as well as practicing using the run and debug function on VSCode. Lab 10 looks into exploring how to use loops to write a program that can take in input, perform an action, and return an output for minimum, maximum, and average. The lab explains how to use Terminal, VSCode, and SNOWBALL. The lab also gives hands-on experience with writing a program to perform these actions.

Purpose

The purpose of this lab is to learn how to program efficiently as well as the use of loops for arrays, along with the commands and practice writing RISC-V utilizing VSCode; this lab also works to use arrays and loops to find minimum, maximum, and average values. This lab is meant to experiment with using RISC-V to take in input and return output with programs and explore different ways to use these commands.

Main Objectives

- Learn how to use loops
- Learn how to use loops for arrays
- Utilize arrays
- Writing code to loop through and print each value
- Writing code to loop through an array to pick up a maximum value
- Writing code to loop through an array to pick up a minimum value
- Writing code to loop through an array to pick up a average value

Brief Explanation

VS Code is an integrated development environment (IDE) where Venus and RISC-V can be accessed and programmed. “RISC-V is an open-source instruction set architecture used to develop custom processors for a variety of applications, from embedded designs to

supercomputers.” according to Synopsys.. “Terminal is an app for advanced users and developers that lets you communicate with the Mac operating system using a command line interface (CLI).” (Apple). Using VSCode, a programmer can use RISC-V to take in inputs, perform operations, and return outputs. Terminal provides a place to run programs that can output errors, what’s causing them, what line, and which column to better assist programmers.

Methodology/Procedure

The procedure starts with writing a program to test and learn more about how the program loops through arrays and prints them. Next, the code is reused to run through the entire array and print only the last value. This program uses this array to teach students what looping in arrays look like and how they work. Reading an array through a loop, calculating values such as minimum, maximum, and average are coded into the program. Finally, these values are calculated using a test and printed with the final values..

Documentation

Code one output:

```
10 20 30 40 50 60
```

Code two output:

```
10 30 60 100 150 210
```

Final code changed to only output final value:

```
  210
```

```
la      t2, min
li      t3, 100000
sw      t3, 0(t2)
```

```
la      t2, max
li      t3, 0
sw      t3, 0(t2)
```

```
la      t2, sum
li      t3, 0
sw      t3, 0(t2)
```

```
la      s0, myarr
la      t1, myarr
li      t2, 6
slli    t2, t2, 2
add     t1, t1, t2
```

```
update_min:
    sw      t1, 0(t2)
```

```
update_max:
    sw      t1, 0(t2)
```

```
.data
myarr:      .word 10, 20, 30, 40, 50, 60
min:        .word 0
max:        .word 0
sum:        .word 0
avg:        .word 0

msg_min:    .asciiz "\nmin value: "
msg_max:    .asciiz "\nmax value: "
msg_avg:    .asciiz "\naverage value: "
```

```
.data
myarr:      .word 90, 120, 10, 55, 60, 25

myarr_end:
sum:        .word 0
min:        .word 0
max:        .word 0
avg:        .word 0
msg_min:    .asciiz "The min value is: "
msg_max:    .asciiz "The max value is: "
msg_avg:    .asciiz "The average value is: "
```

```
The min value is: 10
```

```
The max value is: 90
```

```
.text
main:
    la      s0, myarr
    la      t3, myarr_end
    sub     t3, t3, s0
    li      t4, 4
    div     t3, t3, t4
    mv      t5, t3
```

```
la      t2, sum
lw      t3, 0(t2)
div     t3, t3, t5
la      t2, avg
sw      t3, 0(t2)
```

```
The average value is: 60
```

Questions:

Part One:

1. **How do you know if the series of values is correct?**

   There are two ways to check if the series of values is correct: check the code logic and step through the program. The following screenshots show that there is a loop and points to the next value until it reaches the end–only then does it return the last value. The other way is to use the step through option on VSCode with Run and Debug.

```
addi    s0, s0, 4
la      t1, myarr_end
blt     s0, t1, myloop
```

```
la      t2, temp
lw      t3, 0(t2)
mv      a1, t3
li      a0, 1
ecall
```

```
li      a0, 11
li      a1, 10
ecall
```

2. **How is the address of "myarr_end" different from the address of "temp"?**

   "myarr_end" deals with storing the previous value's address while "temp" deals with storing the value's word in memory.

```
.data
myarr:      .word 10, 20, 30, 40, 50, 60
myarr_end:
temp:       .word 0
```

3. **Before the lw t1, 0(s0) command, which registers, i.e., s0, t0, t1, must be set to a value? If they are not, what will happen? (You can simply explain this.)**

   Before the lw t1, 0(s0), the register that needs to be set to a value is "s0". If the value is not set by "s0", the program will be trying to load from other memory locations that could potentially harm the program's efficiency.

4. **Imagine that a fellow student asks you the following, how would you explain the reason? After the instruction la t2, temp, we have lw t3, 0(t2). Why do we need this second instruction if the t2 already has "temp"?**

t2 is a register that holds the address of temp, while t3 is holding the value stored in temp. We need this second instruction if t2 already has “temp” because we need to read the value that is stored in temp.

```
la      t2, temp
lw      t3, 0(t2)
```

5. **The addi s0, s0, 4 is done intentionally. Why is the "4" there instead of "1"? What happens if the command is addi s0, s0, 1 instead? (Show this as a program, and explain its output.)**

In these statements of “addi”, the last number value in this statement is indicative of the number that needs to be added to the address; if the number is listed as “addi, s0, s0, 4”, then it indicates that there are 4 bytes. If this command is changed to “addi s0, s0, 1”, there is only one byte added to the addresses and would unalign the data.

```
addi    s0, s0, 4
```

```
addi    s0, s0, 1
```

6. **Notice that the loop has the length of 6, and that the "blt" instruction compares s0 to the address after the array. We would do this differently by counting up to 6, that is, counting the number of elements in the array instead of the address where the**

array ends. However, "hard-coding" the array length is a bad idea. How could we use "define/equ/eqv" to code the array length in a better way?

We could use “equ” to code the array length in a better way. According to IBM, “The EQU instruction assigns absolute or relocatable values to symbols. Use it to: Assign single absolute values to symbols.” By using .equ, we can set the length to exactly 6 or whatever length needed for each case and cut off.

```
.equ ARR_LEN, 6
```

Part Two:

1. **Suppose that the array has values 90, 120, 10. Describe (in a step-by-step manner) how a program would examine each one, and update the minimum and/or maximum. When finished, would we have the correct minimum and maximum?**

   If the array with the value 90, 120, and 10 is inputted to this program in a step by step manner, the program would first set 90 to both min and max because it is the first value entered. Then, 12 would become the max value because it is larger than 90, making the max 120 and the min has no change because there hasn’t been a smaller value entered. Finally, when 10 is entered, it is smaller than the current maximum value (120) so max will remain the same; min on the other hand is currently 90, 10 is smaller than 90 and will replace 90 as the min value.

2. **Why are the values 0 and 100,000 not appropriate? Show an example list of numbers that would make these initial values wrong for the minimum, then another example list for the maximum.**

   The values 0 and 100,000 are not appropriate because they set a certain range to the numbers. If, in a scenario where the user has to enter a number larger or smaller than this range, there would be an error. An example list of numbers larger than 100,000 that would make these initial values wrong for the maximum is 100,001, 200,000, and 101,010. An example list of numbers smaller than 100,000 that would make these initial values wrong for the minimum is -1, -3, -10,000.

Part Three:

3. **Suppose that the array has values 90, 120, 10. Describe (in a step-by-step manner) how a program would examine each one, and update the minimum and/or maximum. When finished, would we have the correct minimum and maximum?**

   If the array with the value 90, 120, and 10 is inputted to this program in a step by step manner, the program would first set 90 to both min and max because it is the first value entered. Then, 12 would become the max value because it is larger than 90, making the max 120 and the min has no change because there hasn't been a smaller value entered. Finally, when 10 is entered, it is smaller than the current maximum value (120) so max

will remain the same; min on the other hand is currently 90, 10 is smaller than 90 and will replace 90 as the min value.

Part Four:

1. **Suppose that the array has only the value 90. How should you alter the program to make it work with only one value?**

   The program right now is expecting values to fill in for the minimum and maximum holders. By changing the code to initialize and hold the first element of an array to be set to both the maximum and minimum, the program would work with only one value.

2. **Suppose that the array is empty, that there are no values in it. How should you alter the program to make it still work?**

   If the array is empty with no values in it, there would be an error since the program is expecting values to fill in for the min and max values in order to loop and return the needed values. In order to make the program still work even with an empty array with no values in it, having an edge case by printing a message is one solution.

Part Five:

1. **The average of 10, 11 is reported as 5. Why is it 5, and not 5.5?**

   The average of 10 and 11 is reported as five because this is an integer. An integer is a whole number but not a decimal and this program is meant to return an integer, thus truncating the ".s" from 5.5.

2. **Would the average still work if some of the values are negative? Why or why not?**

   The answer to this question depends on the parameters given to the minimum and maximum values. If it can handle negative numbers, the average would still work if some of the values are negative and stored in their variable. If the negative numbers are not accepted into the proposed range set by the program, the average would not work,

3. **Would the average still work if all of the values are negative? Why or why not?**

   Again, this would depend on the parameters and if the program is accepting negative numbers of that size. The program can stand the chance of adapting negative numbers if the parameters allow it. Another solution is a loop that can check for negative numbers and take its absolute value so that it can still be addressed and returned as the average.

4. **If the array is empty, what is a reasonable output?**

   Going back to the previous part, if the array is empty, there can be an error message returned. Another possibility is returning the number zero as a reasonable output.

5. **Suppose that the array is very long, such as thousands of values. Would the minimum, maximum, and average code still work? Why or why not?**

   Yes, the program can still work because it is built to loop through the array step-by-step. The program would take a long time given the O(n) time complexity, however, it is very possible for these values to still work with minimum, maximum, and average.

6. **Given code that prints an array, it sounds trivial to add code to also find and print the minimum, maximum, and average. How does the original array printing code compare to the final code? Is this what you expected, and why or why not?**

   While printing an array takes a lot of time to step through each number in the array, it would be beneficial to take into account the maximum, minimum, and average. This is more efficient and productive while looping through the array and keeping track of such values for these variables to calculate and return.

Key Code Observations

```
addi    s0, s0, 4
la      t1, myarr_end
blt     s0, t1, myloop

la      t2, temp
lw      t3, 0(t2)
mv      a1, t3
li      a0, 1
ecall

li      a0, 11
li      a1, 10
ecall
```

```
la      t2, min
li      t3, 100000
sw      t3, 0(t2)
```

```
la      t2, max
li      t3, 0
sw      t3, 0(t2)
```

```
la      t2, sum
li      t3, 0
sw      t3, 0(t2)
```

```
la      s0, myarr
la      t1, myarr
li      t2, 6
slli    t2, t2, 2
add     t1, t1, t2
```

```
update_min:
    sw      t1, 0(t2)
```

```
update_max:
    sw      t1, 0(t2)
```

```
.data
myarr:      .word 10, 20, 30, 40, 50, 60
min:        .word 0
max:        .word 0
sum:        .word 0
avg:        .word 0

msg_min:    .asciiz "\nmin value: "
msg_max:    .asciiz "\nmax value: "
msg_avg:    .asciiz "\naverage value: "
```

```
.data
myarr:      .word 90, 120, 10, 55, 60, 25

myarr_end:
sum:        .word 0
min:        .word 0
max:        .word 0
avg:        .word 0
msg_min:    .asciiz "The min value is: "
msg_max:    .asciiz "The max value is: "
msg_avg:    .asciiz "The average value is: "
```

```
.text
main:
    la      s0, myarr
    la      t3, myarr_end
    sub     t3, t3, s0
    li      t4, 4
    div     t3, t3, t4
    mv      t5, t3
```

```
la      t2, sum
lw      t3, 0(t2)
div     t3, t3, t5
la      t2, avg
sw      t3, 0(t2)
```

```
addi    s0, s0, 4
la      t1, myarr_end
blt     s0, t1, myloop
```

```
la      t2, temp
lw      t3, 0(t2)
mv      a1, t3
li      a0, 1
ecall
```

```
li      a0, 11
li      a1, 10
ecall
```

```
.data
myarr:      .word 10, 20, 30, 40, 50, 60
myarr_end:
temp:       .word 0
```

```
la      t2, temp
lw      t3, 0(t2)
```

```
addi    s0, s0, 4
```

```
addi    s0, s0, 1
```

```
.equ ARR_LEN, 6
```